**Diabetes Dataset Predicition**

Context

The objective is to predict based on diagnostic measurements whether a patient has diabetes.

Content

Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

- Pregnancies: Number of times pregnant

- Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test

- BloodPressure: Diastolic blood pressure (mm Hg)

- Pregnancies: Number of times pregnant

- Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test

- BloodPressure: Diastolic blood pressure (mm Hg)

- HeartRate: Heart rate (beats/minute) From Heart Pulse Sensor

- Insulin: 2-Hour serum insulin (mu U/ml)

- BMI: Body mass index (weight in kg/(height in m)^2)

- DiabetesPedigreeFunction: Diabetes pedigree function

- Age: Age (years)

- Outcome: Class variable (0 or 1)

Sources:

(a) Original owners: National Institute of Diabetes and Digestive and

Kidney Diseases

(b) Donor of database: Vincent Sigillito (vgs@aplcen.apl.jhu.edu)

Research Center, RMI Group Leader

The Jupyter Notebook document uses the following libraries:

- Pandas: A library for data manipulation and analysis. It provides data structures and functions for efficiently handling structured data.

- numpy: A library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

- matplotlib: A plotting library for creating static, animated, and interactive visualizations in Python. It provides a MATLAB-like interface for creating a variety of plots, such as line plots, scatter plots, bar plots, histograms, etc.

- seaborn: A data visualization library based on matplotlib. It provides a high-level interface for creating informative and attractive statistical graphics.

- sklearn: A machine learning library that provides a wide range of tools for classification, regression, clustering, and dimensionality reduction. It includes various algorithms, evaluation metrics, and data preprocessing techniques.

- pickle: A module for serializing and deserializing Python objects. It allows you to save trained models or other Python objects to disk and load them back into memory later.

- warnings: A module for issuing warning messages. It can be used to control the display of warning messages during code execution.

Data columns (total 9 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Pregnancies 768 non-null int64

1 Glucose 768 non-null int64

2 BloodPressure 768 non-null int64

3 Insulin 768 non-null int64

4 HeartRate 768 non-null int64

5 BMI 768 non-null float64

6 DiabetesPedigreeFunction 768 non-null float64

7 Age 768 non-null int64

8 Outcome 768 non-null int64

A screenshot of a graph

Description automatically generatedCorrelation between feature with each other :

The correlation matrix shows the correlation coefficients between each feature and the target variable "Outcome". The correlation coefficient ranges from -1 to 1, where -1 indicates a strong negative correlation, 0 indicates no correlation, and 1 indicates a strong positive correlation.

Here are the correlations between each feature and the target variable "Outcome":

- Pregnancies: 0.221898

- Glucose: 0.466581

- BloodPressure: 0.065068

- Insulin: 0.130548

- HeartRate: 0.252841

- BMI: 0.292695

- DiabetesPedigreeFunction: 0.173844

- Age: 0.238356

Based on these correlation coefficients, we can observe that the features "Glucose", "BMI", and "Age" have relatively stronger positive correlations with the target variable "Outcome". This means that as these features increase, the likelihood of having diabetes also tends to increase. On the other hand, the features "BloodPressure" and "DiabetesPedigreeFunction" have weaker positive correlations with the target variable. The feature "Pregnancies" also shows a moderate positive correlation with the target variable.

It's important to note that correlation does not imply causation. These correlation coefficients only indicate the strength and direction of the linear relationship between the features and the target variable. Other factors and variables may also influence the occurrence of diabetes.

Here is another graph instead I giving you all:

A graph with numbers and lines

Description automatically generated

Makin another observation :

# Among the 768 people, 268 people suffer from diabetes and 500 people do not have diabetes.

A graph with blue bars

Description automatically generated

Let’s go through about Ml model:

In machine learning, it is common practice to split the available data into two sets: a training set and a testing set. This is done to evaluate the performance of a machine learning model on unseen data.

The training set is used to train the model, i.e., to estimate the model parameters based on the input features and the corresponding target variables. The model learns patterns and relationships in the training data, which it can then use to make predictions on new, unseen data.

The testing set, on the other hand, is used to evaluate the performance of the trained model. It contains data that the model has not seen during training. By making predictions on the testing set and comparing them to the actual target values, we can assess how well the model generalizes to new, unseen data.

The goal is to build a model that performs well on both the training set and the testing set. If a model performs well on the training set but poorly on the testing set, it is likely overfitting the training data, meaning it is too complex and has learned the noise or random fluctuations in the training data. On the other hand, if a model performs poorly on both the training set and the testing set, it is likely underfitting the data, meaning it is too simple and fails to capture the underlying patterns in the data.

To split the data into training and testing sets, we typically use a random sampling technique. The data is randomly divided into two sets, with a certain percentage allocated to the training set and the remaining percentage allocated to the testing set. The most common split is 70% for training and 30% for testing, but other splits, such as 80-20 or 75-25, can also be used depending on the size of the dataset and the specific problem.

In Python, the `train\_test\_split` function from the `sklearn.model\_selection` module can be used to split the data into training and testing sets. It takes the input features and the target variable as input and returns four sets: the training features, the testing features, the training target variable, and the testing target variable.

Brief About Ml Model used :

Logistic Regression:

Purpose: Used for binary classification tasks, like predicting diabetes based on health measurements.

Algorithm Overview: Models the relationship between input features and the probability of a binary outcome.

Training Process: Estimates coefficients using a labeled dataset, splitting it into training and testing sets.

Prediction Method: Assigns class labels based on the calculated probability exceeding a threshold.

Advantages: Simple, interpretable, efficient, handles numerical and categorical features, provides probabilistic interpretations.

Limitations: Assumes a linear relationship between features and outcomes, may not hold in all cases.

Random Forest Classifier:

Purpose: Handles both classification and regression tasks via ensemble learning.

Algorithm Overview: Constructs multiple decision trees on random data subsets and aggregates predictions.

Strengths: Handles high-dimensional data, captures complex relationships, offers feature importance insights, reduces overfitting risk.

Limitations: Can be computationally expensive, lacks interpretability compared to simpler models.

In essence, Logistic Regression is straightforward, interpretable, and suitable for simpler relationships, while Random Forest Classifier is versatile, robust, and excels in handling complex data but might be more resource-intensive and less interpretable. Both algorithms serve distinct purposes based on the characteristics of the data and the requirements of the task at hand.

**SVM (Support Vector Machine):**

* **Purpose:** SVM is used for both classification and regression tasks, especially in scenarios with a clear margin of separation between classes.
* **Algorithm Overview:** SVM finds a hyperplane that best separates classes in a high-dimensional space, maximizing the margin between them.
* **Training Process:** Identifies the optimal hyperplane using support vectors from the training dataset.
* **Prediction Method:** Classifies new instances based on their position relative to the hyperplane.
* **Advantages:** Effective in high-dimensional spaces, works well with clear margin data, versatile (can use different kernel functions for complex data relationships).
* **Limitations:** Computationally intensive for large datasets, doesn't perform well on noisy datasets, requires proper selection of hyperparameters.

**Gradient Boosting:**

* **Purpose:** Gradient Boosting is used for both regression and classification tasks, building a strong predictive model by combining multiple weak learners.
* **Algorithm Overview:** Builds a series of trees sequentially, with each tree compensating for the errors of the previous ones.
* **Training Process:** Fits new trees to the residuals (errors) of the preceding predictions to correct the model.
* **Prediction Method:** Aggregates predictions from multiple weak learners to create a strong predictive model.
* **Advantages:** Handles complex relationships in data, less prone to overfitting compared to single decision trees, can capture nonlinear patterns.
* **Limitations:** Prone to overfitting with excessive complexity, might require tuning of hyperparameters, computationally expensive compared to simpler models.

Comparison Between Models :

A graph of a graph showing different types of data

Description automatically generated with medium confidence

Conclusion:

In this Jupyter Notebook, we performed various tasks related to data analysis and machine learning. We started by importing the necessary libraries and loading the dataset. We then explored the data by visualizing it and analyzing its statistical properties. After that, we preprocessed the data by handling missing values, encoding categorical variables, and splitting the dataset into training and testing sets.

Next, we trained and evaluated different machine learning models on the dataset. We used logistic regression, random forest, support vector machine, and gradient boosting classifiers. We compared the performance of these models using accuracy, classification report, and AUC-ROC metrics.

Observations:

1. The logistic regression model achieved an accuracy of 77% on the resampled data.

2. The random forest model outperformed other models with an accuracy of 83% on the resampled data.

3. The support vector machine model had an accuracy of 67% on the resampled data.

4. The gradient boosting model achieved an accuracy of 70% on the resampled data.

5. The bar chart comparing the accuracies of different models showed that the random forest model had the highest accuracy.

Overall, the random forest model performed the best among the models evaluated in this notebook. It achieved the highest accuracy and showed promising results for predicting the outcome of diabetes based on the given diagnostic measurements. Further optimization and fine-tuning of the models could potentially improve their performance.

Sampel Web app:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Precision: It measures the accuracy of the positive predictions made by the model. It's the ratio of correctly predicted positive observations to the total predicted positives, i.e., TP / (TP + FP), where TP is true positives and FP is false positives.
2. Recall (Sensitivity or True Positive Rate): It measures the ability of the model to correctly identify positive instances. It's the ratio of correctly predicted positive observations to the all actual positives, i.e., TP / (TP + FN), where FN is false negatives.
3. F1-score: It's the harmonic mean of precision and recall. F1-score considers both precision and recall and provides a single score that balances both. It's calculated as 2 \* (precision \* recall) / (precision + recall).
4. Support: The number of actual occurrences of the class in the dataset.

Here is classification report

RandomForestClassifier(random\_state=42)

precision recall f1-score support

0 0.64 1.00 0.78 99

1 0.00 0.00 0.00 55

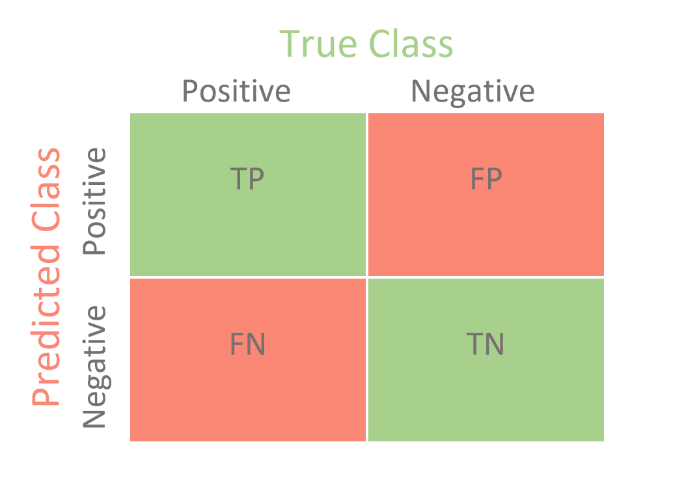
accuracy 0.64 154

macro avg 0.32 0.50 0.39 154

weighted avg 0.41 0.64 0.50 154

A graph of a number of different colored bars

Description automatically generated

A confusion matrix is a table used to evaluate the performance of a classification algorithm. It presents a summary of the predictions versus the actual ground truth across different classes. It consists of four important metrics:

* True Positives (TP): Instances where the model correctly predicts the positive class.
* True Negatives (TN): Instances where the model correctly predicts the negative class.
* A diagram of a confusion matrix

  Description automatically generatedFalse Positives (FP): Instances where the model predicts the positive class incorrectly.
* False Negatives (FN): Instances where the model predicts the negative class incorrectly.

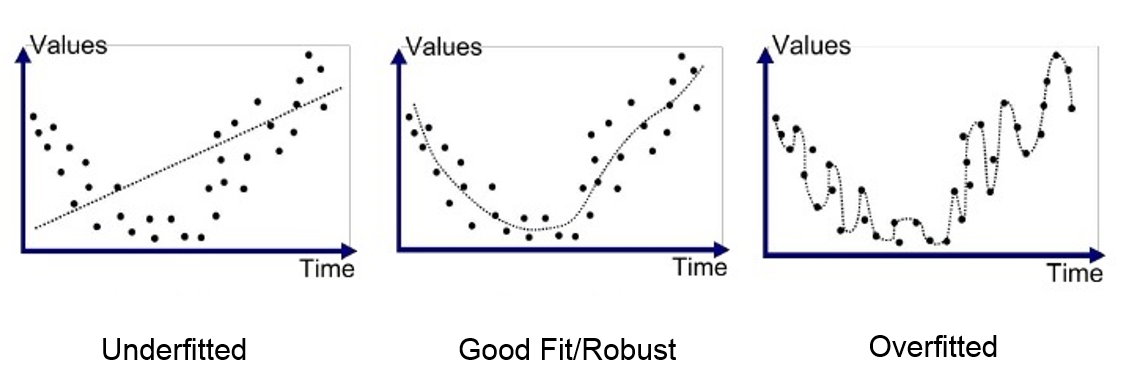
**Project Overview:**

* **Technology Used:** Flask (Python web framework), **pickle** for model persistence, and a chatbot interface.
* **Objective:** Predict diabetes levels and provide a chatbot interface for user interaction.

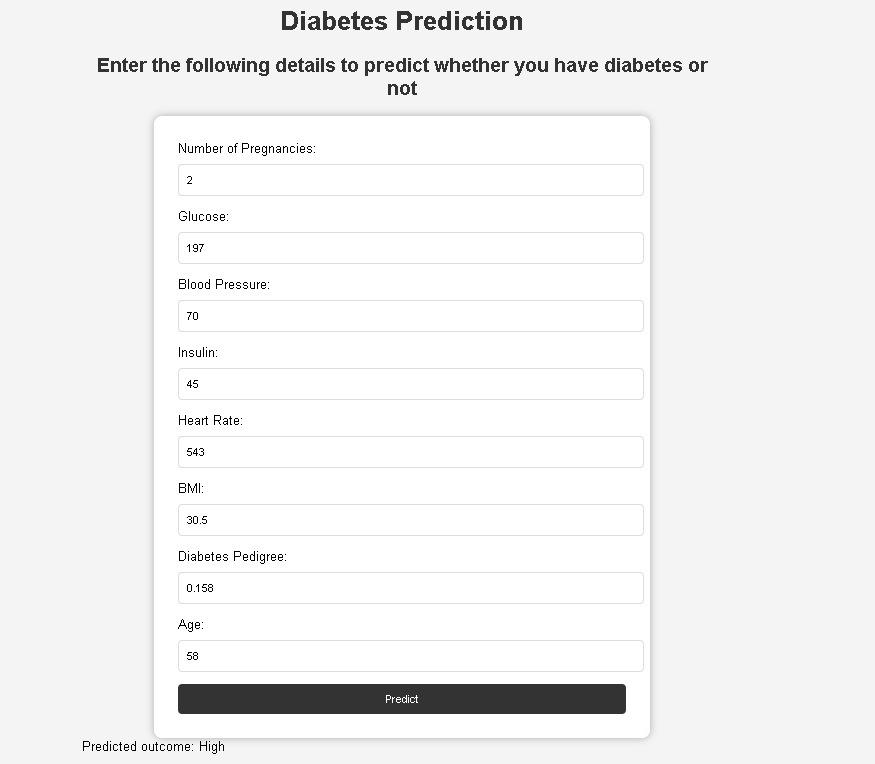
**Key Components:**

1. **Flask Application Setup:** Configures Flask to handle HTTP requests and responses.
2. **Model Loading:** Loads a pre-trained machine learning model (**modelbest.pkl**) for diabetes prediction.
3. **Endpoints:**
   * **/**: Renders an HTML template for the user interface.
   * **/predict**: Accepts input data in JSON format, processes predictions using the loaded model, and returns predictions as JSON.
4. **Chatbot Integration:** Includes a chatbot interface for user interaction, possibly for inquiries, assistance, or guiding users through the prediction process.
5. **Prediction Logic:** Parses input data, processes it, and uses the model to predict diabetes levels based on health-related features.
6. **Output:** Provides a prediction result indicating either 'High', 'Low', or 'Invalid Prediction' for diabetes levels.

**Purpose:** This project aims to offer a dual functionality: predicting diabetes risk based on health parameters and facilitating user interaction through a chatbot interface. By integrating a chatbot, it enhances user experience, allowing for queries, assistance, and guidance throughout the prediction process, thereby promoting user engagement and understanding of health-related predictions.

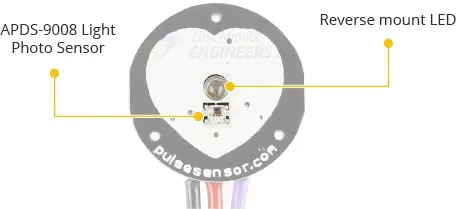


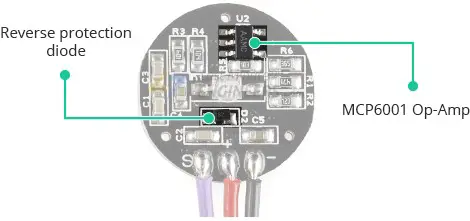
Sample With Newdata :



Monitor the Heart Rate using Pulse Sensor and Arduino

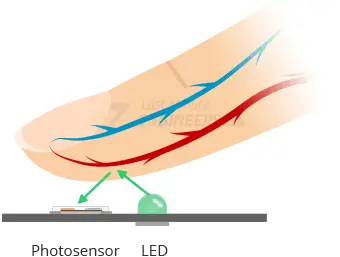
Hardware Overview

The front of the sensor, with the heart logo, is where you put your finger. You’ll also notice a tiny circular opening through which the Kingbright’s reverse mounted green LED shines.

Just beneath the circular opening is a small ambient light photo sensor – APDS-9008 from Avago. This sensor is similar to the ones used in cell phones, tablets, and laptops to adjust the screen’s brightness based on the ambient lighting conditions.

On the back of the module are an MCP6001 Op-Amp from Microchip and a few resistors and capacitors that make up the R/C filter network. Additionally, there is a reverse protection diode to prevent damage in the event that the power leads are accidentally reversed.

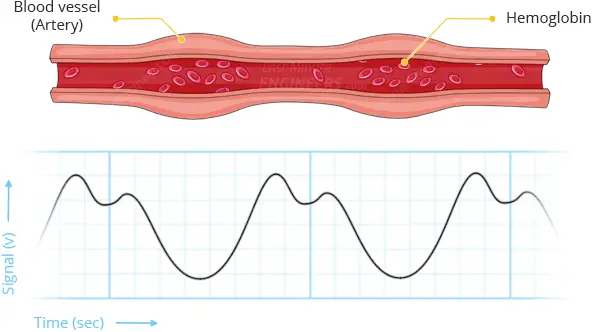
The module requires a DC power supply ranging from 3.3 to 5V and draws less than 4mA of current.

How Does a Pulse Sensor Work?

The theory behind optical heart-rate sensors is very simple. If you’ve ever shined a flashlight through your fingers and observed your heartbeat pulsing, the concept of optical heart-rate pulse sensors can be easily grasped.

A pulse sensor, like any other optical heart-rate sensor, works by shining a green light (~ 550nm) on the finger and measuring the amount of reflected light with a photosensor.

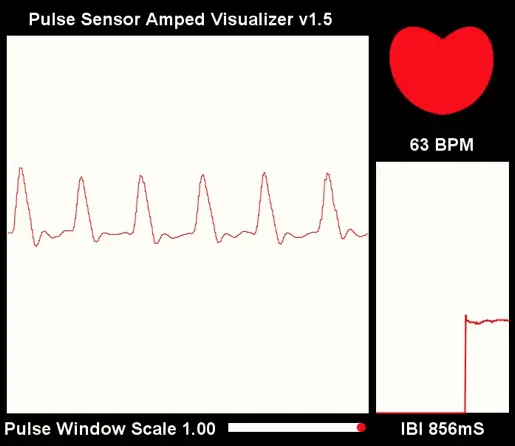
This optical pulse detection technique is known as a Photoplethysmogram.



The oxygenated hemoglobin in arterial blood has the property of absorbing green light. The redder the blood (the higher the hemoglobin), the greater the absorption of green light. With each heartbeat, blood is pumped through the finger, causing a change in the amount of reflected light, which in turn produces a waveform at the photosensor’s output.

As you keep shining light and taking photosensor readings, you quickly begin to obtain a heart-beat pulse reading.

This signal from the photosensor is typically small and noisy; therefore, it is passed through an R/C filter network and then amplified with an Op-Amp to create a signal that is significantly larger, cleaner, and easier to detect.

Processing Visualizer

The Pulse Sensor developers have created software to visualize the Pulse Sensor data on your computer. It is written in the Processing programming language. This software displays all of the data that the Arduino receives from the Pulse Sensor. It plots the user’s heart rate in real time. It also displays the BPM (Beats Per Minute) and plots IBI (Interbeat Interval) over time.

This Processing sketch does not perform any calculations! They are all done on the Arduino board, so to use the visualizer, you must have an Arduino running the PulseSensor\_BPM sketch. This software simply reads the Serial Port and visualizes the data.

Connection Arduino with Heart Pulse Sensor :

A diagram of a sensor pinout

Description automatically generated

